รีเคอร์ชัน และสเปรดชีต

บุญเลิศ เอี่ยมทัศนา

มีผู้ถามผมว่ามีเหตุผลใดที่ต้องเขียนโปรแกรมในแบบเรียกตนเองและเมื่อไรไม่ต้อง หรือไม่ก็ถามในทำนองที่ว่า จะทราบได้อย่างไรว่าโปรแกรมนี้จะต้องเขียนในแบบเรียกตนเองและจากเรื่อง "การเขียนโปรแกรมในยุคที่สาม" ได้ค้างเรื่องการเขียนโปรแกรมในแบบสเปรดชีตไว้ เพื่อนำมาเป็นตัวอย่างในบทความนี้ จึงขอนำมาเขียนรวมกัน

โปรแกรมเรียกตนเอง (recursion)

ผมคิดว่าเรื่องแรกควรจะทำความเข้าใจกับเรื่องราวของโปรแกรมเรียกตนเองก่อนคือ โปรแกรมเรียกตนเองเขียนได้ในบางภาษาเท่านั้น คือ ภาษาที่กำหนดให้เขียนโปรแกรมย่อยเป็นบล็อก เช่น ภาษาปาสคาล ซี โคบอล รวมทั้งภาษาแอสเซมบลี และเขียนไม่ได้ (เขียนได้ยากมากจนถือว่าเขียนไม่ได้) ในบางภาษาเช่น เบสิก ฟอร์แทรน ส่วนภาษาที่ต้องเขียนโปรแกรมในแบบเรียกตนเองอย่างเป็นธรรมดาได้แก่ ภาษาลิสป์ และโปรล็อก

โปรแกรมเรียกตนเองเกือบทั้งหมดเป็นโปรแกรมย่อย (ในภาษาซี ให้โปรแกรมหลักเรียกตนเองได้ แต่ไม่ค่อยมีที่ใช้) มีลักษณะคล้ายกับวงรอบ WHILE กล่าวคือ เมื่อเราเขียนวงรอบ WHILE ว่า

```
WHILE เงื่อนไขเป็นจริง DO
    ถ้อยแถลง 1;
    . .
```

ถ้อยแถลง n;
ENDWHILE

นั่นคือ ถ้าเงื่อนไขเป็นจริงจะทำงานในวงรอบ WHILE คือ ทำถ้อยแถลง 1 ถึง n ซ้ำอยู่เช่นนี้ จนกว่าเงื่อนไขเป็นเท็จ (จะต้องมีถ้อยแถลงหนึ่งที่ทำให้เกิดการเปลี่ยนแปลงเงื่อนไข) เราสามารถเปลี่ยนถ้อยแถลง WHILE เป็นใช้ IF กับ GOTO ได้คือ

```
START: IF เงื่อนไขเป็นจริง THEN
         ถ้อยแถลง 1;
         ถ้อยแถลง n;
         GOTO START
FIN: ENDIF
```

คำว่า START: และ FIN: เป็นจุดอ้างอิงในการกระโดด (label) จะเห็นว่าทำงานเหมือนวงรอบ WHILE เช่นกัน คือ ถ้าเงื่อนไขเป็นจริงก็จะทำถ้อยแถลง 1 ถึง n แล้วกระโดดไปที่ START: คือ ตรง IF ทดสอบเงื่อนไขใหม่ หากเงื่อนไขเป็นเท็จจะไม่ทำถ้อยแถลง IF เท่ากับกระโดดไปที่ FIN: คือ ENDIF รูปแบบนี้ คือ รูปแบบที่ใช้ในการเขียนโปรแกรมเรียกตนเอง จะยกตัวอย่างประกอบกับกรณีที่ควรหรือไม่ควรเขียนเป็นโปรแกรมเรียกตนเองต่อไป

เมื่อไม่ควรเขียนเป็นโปรแกรมเรียกตนเอง

สมมติว่าเราต้องการนำอักขระในสตริง ซึ่งกำหนดให้ปิดท้ายด้วยเครื่องหมาย $ มาดำเนินการใดก็ได้ทีละตัว ซึ่งในที่นี้ เพื่อเป็นตัวอย่างจึงเพียงแต่ให้นำออกจอภาพ จะเขียนโปรแกรมในภาษาปาสคาลได้ ดังโปรแกรมที่ 1

โปรแกรมนี้จะพิมพ์อักษร a ถึง f ออกทางจอภาพ โดยไม่พิมพ์เครื่องหมาย $ ซึ่งเราสามารถเขียนเป็นโปรแกรมเรียกตนเองได้ดังโปรแกรมที่ 2

โปรแกรมนี้จะพิมพ์ภาพอักษร a ถึง f ออกจอภาพเช่นเดียวกับที่เราใช้วงรอบ WHILE จะเห็นว่าซอร์สโค้ดยาวกว่าและเมื่อคอมพายล์แล้ว จะได้ออบเจคโค้ดยาวกว่าเช่นกัน ทำให้ใช้เวลาในการรันมากกว่า และนอกจากนี้ยังต้องบวกเวลาที่เสียไปในการเรียกโปรแกรมย่อย (call) คือที่เรียกตนเอง และเวลาที่เสียไปในการส่งกลับจากโปรแกรมย่อย (return) จึงเป็นกรณีที่ไม่ควรเขียนเป็นโปรแกรมเรียกตนเองเป็นอย่างยิ่ง (และก็ไม่มีใครเขียนกัน)

โปรแกรมที่ 1

```
PROGRAM  Example11(input,output);
VAR    astr : String[30];
       i    : Integer;

BEGIN
  astr := 'abcdef$';
  i := 1;
  WHILE  astr[i] <> '$'  DO
    BEGIN
    Write(astr[i]);
    i := i + 1;
    END;
END.
```

โปรแกรมที่ 2

```
PROGRAM  Example12(input,output);
VAR    astr : String[30];
       i    : Integer;

PROCEDURE  PrintChar;
BEGIN
  IF  astr[i] <> '$'  THEN
    BEGIN
    Write(astr[i]);
    i := i + 1;
    PrintChar;
    END;
END;

BEGIN    { main }
  astr := 'abcdef$';
  i := 1;
  PrintChar;
END.
```

กรณีที่ต้องพิจารณา

เช่นกรณีที่เราอ่านข้อมูล (จากไฟล์ในดิสก์หรือคีย์บอร์ด หรืออื่นๆ) เข้ามาดำเนินกรรมวิธีโดยที่เราไม่ทราบว่าจำนวนข้อมูลจะมีมากน้อยเท่าไร เสร็จแล้วเราต้องการข้อมูลนี้มาดำเนินกรรมวิธีย้อนหลัง คือจากข้อมูลสุดท้ายมายังข้อมูลแรก

ตัวอย่างเช่น ให้ผู้ใช้ป้อนข้อมูลเป็นอักษรภาษาอังกฤษตัวเล็ก เราอ่านเข้ามาดำเนินกรรมวิธี คือ แปลงเป็นอักษรใหญ่เก็บไว้ จนกว่าผู้ใช้จะกดคีย์ . (จุด) นำข้อมูลมาดำเนินกรรมวิธีย้อนหลังคือ ให้แสดงภาพอักษรใหญ่นี้ทางจอภาพ จากอักษรตัวสุดท้ายย้อนกลับไปหาตัวแรก จะเขียนโปรแกรมโดยใช้วงรอบ WHILE ได้ดังโปรแกรมที่ 3

โปรแกรมนี้จะดำเนินการตามที่กล่าวมา ขอให้สังเกตด้วยว่า ผมใช้วงรอบ FOR ต่อกับ WHILE เพราะเราทราบจำนวนอักขระที่ป้อนแน่นอนแล้ว ในตอนนี้คือค่าที่อยู่ใน I ก่อนที่ผมจะเขียนโปรแกรมเรียกตนเอง ขอทำความเข้าใจในเรื่องการเรียกโปรแกรมย่อยก่อนคือ

ถ้อยแถลง 1;
ถ้อยแถลง 2;

Xray ;
ถ้อยแถลง 3;
ถ้อยแถลง 4;

โปรแกรมนี้จะทำถ้อยแถลง 1 กับ 2 เสร็จแล้ว จะเรียกใช้โปรแกรมย่อย Xray ทำงานในโปรแกรมย่อย Xray จนจบ แล้วส่งกลับมายังโปรแกรมนี้ตรงถ้อยแถลง 3 เพื่อทำถ้อยแถลง 3 และ 4 ตรงนี้ดูเหมือนไม่สำคัญ แต่จะสำคัญมากเมื่อเขียนโปรแกรมในแบบเรียกตนเอง จากตัวอย่างที่ยกมาจะเขียนเป็นโปรแกรมเรียกตนเองได้ดังโปรแกรมที่ 4

จะเห็นว่าเขียนโปรแกรมได้สั้นกว่ากัน เมื่อคอมพายล์แล้ว ก็สั้นกว่ากันด้วย แต่เวลาในการรันอาจไม่เร็วกว่า เพราะต้องเสียเวลาในการเก็บค่าตัวแปร ch ลงสแต็ก เสียเวลาในการเรียกโปรแกรมย่อย (คือที่เรียกตนเอง) และการส่งกลับ เมื่อรวมแล้วอาจมากกว่าที่ใช้วงรอบ WHILE กับ FOR

สำหรับการทำงานของโปรแกรมนี้ เริ่มด้วยการอ่านคีย์ สมมติว่าผู้ใช้กดคีย์ a ก็จะนำออกแสดงให้ผู้ใช้เห็น แล้วตรวจดูว่าเป็นจุดหรือไม่ ถ้าไม่เป็นจุด จะเปลี่ยนค่าเป็น A ให้แก่ตัวแปร ch เท่ากับเป็นการเก็บค่า A ลงสแต็ก เพราะภาษาที่ให้เขียนโปรแกรมเรียกตนเองได้ จะเก็บค่าของตัวแปรที่เป็นโลคอล (local) ไว้ในสแต็ก เสร็จแล้วเรียกตนเองทำซ้ำเดิมอีก คือเริ่มด้วยการอ่านค่าคีย์ ตรงนี้เทียบได้กับวงรอบ WHILE

สมมติว่าผู้ใช้ป้อน a, b, c, d และจุด ตอนนี้ค่าใน ch เป็นจุด จึงไม่ทำประโยค IF มาทำประโยคที่ผมให้พิมพ์วรรค แล้วจบโปรแกรมย่อย ต่อไปเป็นการส่งกลับ ซึ่งจะถอนค่า ch จากสแต็ก ค่า ch ในตอนนี้จึงเป็นอักขระตัวสุดท้าย คือ D การส่งกลับจะกลับไปหลังประโยคที่เรียกตนเอง คือ Write(ch) เทียบได้กับถ้อยแถลง 3 ที่กล่าวมา พิมพ์อักษร D ออกจอภาพ พิมพ์วรรคอีก ส่งกลับอีก จะเป็นการพิมพ์อักษร C, B และ A ออกจอภาพตามลำดับ เสร็จแล้วจะกลับไปยังโปรแกรมที่เรียกใช้ Recursion 2 คือโปรแกรมหลัก

จะเห็นว่าโปรแกรม Recursion 2 เทียบได้กับการใช้วงรอบ WHILE ต่อด้วยวงรอบ FOR สำหรับถ้อยแถลงที่ให้พิมพ์วรรคเพิ่มเข้ามาเพื่อความสะดวกในการอธิบายเท่านั้น ขอให้ทดลองรันโดยตัดประโยคนี้ออกครับ

ในการพิจารณาว่าในกรณีนี้จะเขียนโปรแกรมเรียกตนเองหรือไม่ ขอให้ดูในโปรแกรมที่ใช้วงรอบ WHILE จะเห็นว่าเราเก็บข้อมูลไว้ในอะเรย์ barray การใช้อะเรย์แบบนี้เรียกว่า ใช้อะเรย์อย่างสแต็ก (แต่ไม่ใช่สแต็กของซีพียูอย่างเช่นในโปรแกรมเรียกตนเอง) หากเราทราบหรือสามารถกำหนดจำนวนข้อมูลสูงสุดที่จะป้อนได้และไม่มากนัก ควรใช้วงรอบ WHILE ดังเช่นโปรแกรม MicroCal ในเรื่อง **"การเขียนโปรแกรมในยุคที่สาม"** (ไมโครฯ 36)

แต่การใช้วงรอบ WHILE กับสแต็กเทียมดังกล่าวนี้จะเปลืองเนื้อที่ในหน่วยความจำ หากข้อมูลที่จะให้เก็บมีหลายตัว มีเป็นจำนวนมาก หรือไม่อาจคาดได้เลยว่าจะมีประมาณเท่าใด ก็ควรจะเขียนเป็นโปรแกรมเรียกตนเอง (แต่ก็ต้องระลึกไว้ด้วยว่าสแต็กของซีพียูก็มีขนาดจำกัดเช่นกัน) หากไม่มีปัญหาเรื่องหน่วยความจำแล้ว ไม่ควรเขียนเป็นโปรแกรมเรียกตนเอง ทางออกอีกทางหนึ่งเมื่อจำนวนของข้อมูลไม่แน่นอน คือ ใช้การกำหนดตัวแปรแบบไดนามิกส์ เช่น Pointer ในภาษาปาสคาล

อีกรูปแบบหนึ่ง

โปรแกรมเรียกตนเองตามที่ยกมาเป็นตัวอย่าง อยู่ในแบบที่ว่าจะเรียกตนเองเมื่อเงื่อนไขนั้นเป็นจริง และไม่ได้ให้ทำ

โปรแกรมที่ 3

```
PROGRAM  Example21(input,output);
VAR    barray : ARRAY[1..30] OF Char;
       i : Integer;  ch : Char;

BEGIN
  i := 0;
  Read(KBD,ch);  Write(ch);
  WHILE  ch <> '.'  DO
    BEGIN
    i := i + 1;
    barray[i] := UpCase(ch);
    Read(KBD,ch);  Write(ch);
    END;
  FOR  i := i DOWNTO 1  DO
    Write(barray[i]);
END.
```

โปรแกรมที่ 4

```
PROGRAM  Example22(input,output);

PROCEDURE  Recursion2;
VAR    ch : Char;
BEGIN
  Read(KBD,ch);  Write(ch);
  IF  ch <> '.'  THEN
    BEGIN
    ch := UpCase(ch);
    Recursion2;
    Write(ch);
    END;
  Write(' ');
END;

BEGIN  { main }
  Recursion2;
END.
```

อะไรเมื่อเงื่อนไขเป็นเท็จ หากเราประสงค์ที่จะให้ดำเนินการอย่างหนึ่งอย่างใดเมื่อเงื่อนไขเป็นเท็จ (และไม่เรียกตนเอง) อาจเขียนโปรแกรมได้ในรูปแบบดังนี้ คือ

```
PROCEDURE CallSelf;
BEGIN
  IF เงื่อนไขเป็นจริง THEN
  BEGIN
  ถ้อยแถลง 1;
  ถ้อยแถลง n;
  CallSelf;
  END
ELSE
  BEGIN
  ถ้อยแถลง a;
    .
    .
  ถ้อยแถลง m;
  END;
END;
```

แต่ที่นิยมเขียนกัน จะขอยกตัวอย่างเช่นเดิมคือ ให้ผู้ใช้ป้อนอักษรตัวเล็กปิดท้ายด้วยจุด ซึ่งโปรแกรมจะแสดงเป็นอักษรใหญ่ย้อนกลับ แต่จุดซึ่งเดิมไม่ได้ให้ทำอะไร จะให้แสดงเครื่องหมาย # ออกจอภาพ จะเขียนเป็นโปรแกรมได้ดังโปรแกรมที่ 5 จะเห็นว่ารูปแบบของโปรแกรมกลายเป็นว่าให้เรียกตนเองเมื่อเงื่อนไขเป็นเท็จ และยุติการเรียกตนเองเมื่อเงื่อนไขเป็นจริง (การคิดในลักษณะนี้ใช้ในภาษาโปรล็อก)

ดังนั้น โดยทั่วไปจึงไม่กล่าวว่าโปรแกรมเรียกตนเองมีลักษณะเป็นวงรอบ WHILE และจะกล่าวว่าเป็นวงรอบ UNTIL ก็ไม่ได้ เพราะวงรอบ UNTIL ไม่ตรวจสอบเงื่อนไขก่อน จึงกล่าวแต่เพียงว่า โปรแกรมเรียกตนเองต้องมีจุดจบที่แน่นอนคือ ต้องมีเงื่อนไขที่ไม่เรียกตนเองอย่างน้อย 1 กรณี แต่ผมคิดว่าหากคิดถึงวงรอบ WHILE จะทำให้เข้าใจโปรแกรมเรียกตนเองดีขึ้น

กรณีที่ควรเขียนเป็นโปรแกรมเรียกตนเอง

ความจริงแล้ว ควรจะกล่าวว่าเป็นกรณีที่ต้องเขียนเป็นโปรแกรมเรียกตนเอง แต่เพราะว่าโปรแกรมเหล่านี้ สามารถเขียนในรูปแบบธรรมดาได้โดยใช้สแต็กช่วย จึงใช้คำว่าควร แต่ก็มีข้อพิจารณาว่า ถ้าเขียนในรูปแบบธรรมดาจะเขียนโปรแกรมได้ยากกว่ายาวกว่า ดังนั้น ในภาษาที่เขียนโปรแกรมในแบบเรียกตนเองได้ จะเขียนในแบบเรียกตนเอง อย่างน้อยที่สุดก็ในครั้งแรก เมื่อโปรแกรมทำงานอาจเปลี่ยนเป็นโปรแกรมธรรมดาหากเห็นว่าเหมาะสมกว่า

โปรแกรมที่ 5

```
PROGRAM  Examplenn(input,output);

PROCEDURE  Recursion2;
VAR    ch : Char;
BEGIN
  Read(KBD,ch);  Write(ch);
  IF  ch = '.' THEN
    Write('#')
  ELSE
    BEGIN
    ch := UpCase(ch);
    Recursion2;
    Write(ch);
    END;
  Write(' ');
END;

BEGIN  { main }
  Recursion2;
  Write('!');
END.
```

จะขอยกตัวอย่างโปรแกรมในแบบนี้ก่อนว่ามีลักษณะเช่นไร แล้วจึงจะเรียนให้ทราบว่าจะอยู่ในงานใดบ้าง ก็ต้องขอให้พลิกไปดูเรื่อง "การเขียนโปรแกรมในยุคที่สาม" ในฉบับนี้ก่อนครับ โดยเฉพาะความหมายของคำว่า Infix กับ Postfix และที่ผมกล่าวไว้ว่า นำไปทำเป็นโปรแกรมในแบบสเปรดชีตได้ บังเอิญโปรแกรมนี้เหมาะที่จะเขียนในแบบเรียกตนเอง ผมจึงโอนมากล่าวในบทความนี้

โปรแกรมในแบบสเปรดชีตมีเรื่องราวอยู่หลายเรื่อง เรื่องหนึ่งที่ผมจะขอนำมาพูดคือ โปรแกรมในแบบนี้เสมือนตีตารางกระดาษเป็นช่องๆ เรียกว่า เซลล์ (cell) ในแต่ละเซลล์นี้ให้ผู้ใช้ป้อนข้อมูลได้แบบหนึ่งแบบใด ดังต่อไปนี้เช่น

(0) apple

(1) 35.5

(2) 20 + 22.7 * 55

(3) 39 – B8 + 75 * C5

คือ (0) ผู้ใช้จะป้อนเป็นข้อความใดก็ได้ หรือ (1) จะป้อนเป็นค่าคงตัวก็ได้ หรือ (2) จะป้อนเป็นข้อความแสดงถึงการคำนวณของค่าคงตัวหลายๆค่าก็ได้ หรือ (3) แสดงถึงการคำนวณของค่าคงตัวและตัวแปรหลายๆค่าก็ได้

โดยที่เราให้ความหมายไว้ว่า เช่น เราแบ่งกระดาษเป็น 10×10 เซลล์ ดังแสดงในรูปที่ 1 เราจะให้ผู้ใช้อ้างถึงเซลล์แต่ละเซลล์ ซึ่งหมายถึงว่าเป็นชื่อเซลล์ด้วยได้ดังนี้ คือ A0 ถึง A9, B0 ถึง B9 และต่อๆไปจนถึง J0 ถึง J9 ดังนั้นที่เขียนไว้ในแบบที่ 3 จะหมายความว่า

"ให้ 39 เป็นตัวตั้ง ลบด้วยค่าในเซลล์ B8 แล้วนำไปบวกกับค่าที่เกิดจากผลคูณของ 75 กับค่าในเซลล์ C5"

ในการใช้งานสเปรดชีตแบบนี้ ผู้ใช้จะมองชื่อเซลล์ประหนึ่งว่าเป็นตัวแปร และผมจะเรียกว่าเป็นตัวแปรในโปรแกรมด้วย ผมจะไม่กล่าวถึงโปรแกรมที่ให้ผู้ใช้ป้อนค่าไปเก็บไว้ในเซลล์ต่างๆ ซึ่ง

	A	B			I	J
0						
1						
2						
3						
4						
5						
6						
7						
8						
9						

รูปที่ 1 ลักษณะของสเปรดชีตเปรียบเสมือนแบ่งหน่วยความจำออกเป็นตาราง

เราจะเก็บไว้เป็นสตริงตรงตามที่ผู้ใช้ป้อน เช่นในแบบที่ (2) จะเก็บเป็นสตริง "20 + 22.7 * 55"

จุดที่ผมจะนำมากล่าวถึงคือ เมื่อผู้ใช้ต้องการให้แสดงค่าในเซลล์ (เฉพาะค่าที่เป็นตัวเลขและการคำนวณ) เช่นถ้าผู้ใช้เก็บค่าที่ยกมาเป็นตัวอย่างทั้ง 4 แบบไว้ในเซลล์ A0 ถึง A3 ตามลำดับ เราอาจให้ผู้ใช้ใช้คำสั่งว่า Display (A1) เครื่องก็จะแสดงค่า 33.5 ออกมา ถ้าใช้คำสั่งว่า Display (A2) เครื่องจะแสดงค่า 97.7 ออกมา ซึ่งเราจะเขียนโปรแกรมได้ดังโปรแกรมที่ 6

จะเห็นว่าเราเขียนโปรแกรมได้ง่ายๆ โดยใช้วงรอบ WHILE ซึ่งขออธิบายเพิ่มดังนี้ ผมเพียงแค่เขียนบางส่วนของโปรแกรมในเรื่องนี้ การกำหนดค่าต่างๆที่เห็นอยู่ก่อนถึงโพรซีเยอร์ Display คือ ค่าที่กำหนดไว้เป็นส่วนกลาง สำหรับใช้กับส่วนอื่นๆของโปรแกรมด้วย

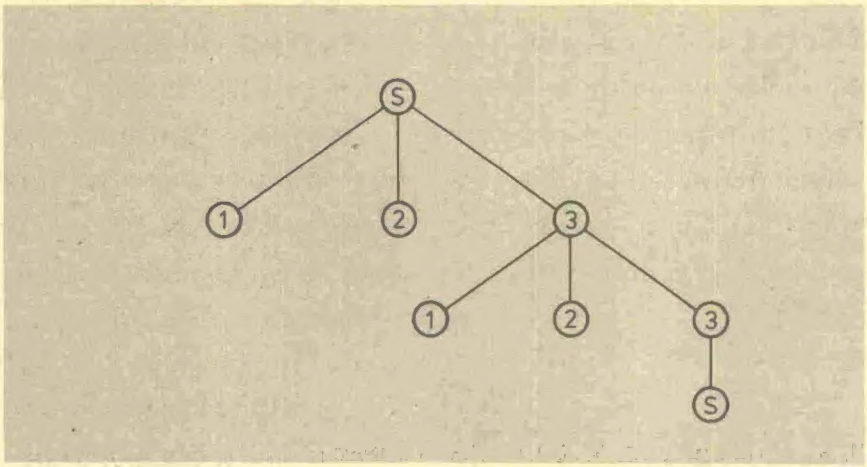
โพรซีเยอร์ Display เพียงแต่เตรียม SP (stack pointer) แล้วเรียกใช้ Calculate ซึ่งจะดำเนินการดังที่จะกล่าวต่อไป ได้ผลอยู่ที่ชั้นยอดของสแต็ก ซึ่งในกรณีนี้จะเป็นชั้นล่างสุดคือชั้นที่ 0 ซึ่ง Display นำมาแสดงออกจอภาพ

สำหรับโพรซีเยอร์ Calculate จะนำค่าที่ผ่านมา ซึ่งเป็นชื่อของเซลล์เช่น H5 คือเป็นสตริงที่มีอักษร 1 ตัวและตัวเลข 1 ตัว มาแยกเป็น row และ col และให้ค่าสตริงที่เก็บในเซลล์นี้แก่ตัวแปร astr เสร็จแล้วเปลี่ยนข้อมูลใน astr เป็นแบบ Postfix ด้วย ConvToPostfix (ไม่ได้แสดงโปรแกรมนี้) ให้แก่ outstr

แบบของ outstr ในโปรแกรมนี้แตกต่างกับในเรื่องการเขียนโปรแกรมในยุคที่สาม คือเป็นอะเรย์ของสตริง 12 ในวงรอบ WHILE ได้ตั้งเงื่อนไขไว้ว่าถ้าโทเค็นไม่ใช่ $ ให้ทำงานในวงรอบ

```
(*  part of program spread sheet  *)

CONST  MAXTOK = 80;
TYPE   str2 = String[2];
       str12 = String[12];
       str80 = String[80];
       outstrT = ARRAY[1..MAXTOK] OF str12;
       rowT = 'A'..'J';  colT = '0'..'9';
VAR    sheet : ARRAY[rowT,colT] OF str80;

PROCEDURE  Display(cell : Str2);
VAR    SP :Integer;
       row : rowT;  col : colT;
       stack : ARRAY[0..20] OF Real;
       token : Str12;
       ch : Char;

  PROCEDURE  Calculate(cell : Str2);
  VAR  astr : Str80;
       outstr : OutstrT;
       i : Integer;
  BEGIN
    row := cell[1];  col := cell[2];
    astr := sheet[row,col];
    ConvToPostfix(astr, outstr);  i := 1;
    token := outstr[i];  i := i + 1;
    WHILE  token <> '$'  DO
      BEGIN
      ch := token[1];
      IF  ch IN ['0'..'9','-']  THEN
        Push(token)
      ELSE IF  ch IN ['+','-','*','/','^']  THEN
        call(ch);
      token := outstr[i];  i := i + 1;
      END;
  END;

BEGIN  { Display }
  SP := 0;
  Calculate(cell);
  WriteLn(stack[0]);
END;
```

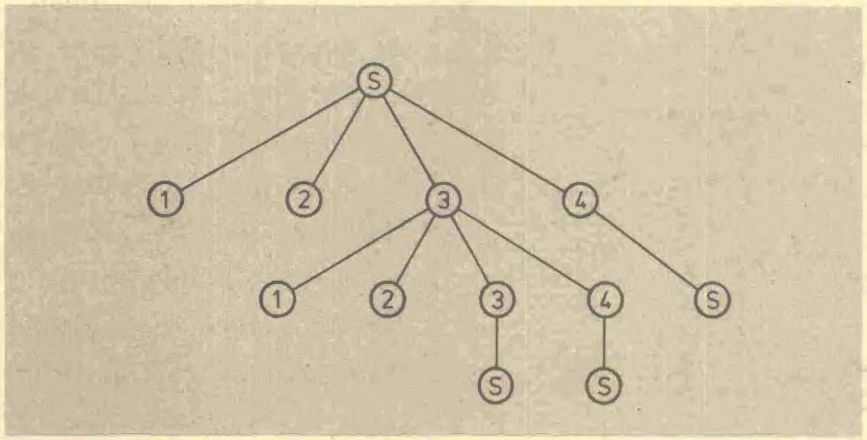
โปรแกรมที่ 6 แสดงส่วนหนึ่งของโปรแกรมในแบบสเปรดชีต



รูปที่ 2 แสดงโครงสร้างข้อมูลในแบบสเปรดชีต อักษร S คือเริ่มต้นและที่ห้อยไว้หลังหมายเลข 3 หมายถึงซ้ำเก่า คือจะซ้อนๆกันไปกี่ชั้นก็ได้ เป็นแบบต้นไม้ [tree] ที่ไม่รู้จบ

```
PROCEDURE  Calculate(cell : Str2);
VAR  astr : Str80;
     outstr : OutstrT;
     i : Integer;
BEGIN
  row := cell[1];  col := cell[2];
  astr := sheet[row,col];
  ConvToPostfix(astr, outstr);  i := 1;
  token := outstr[i];  i := i + 1;
  WHILE  token <> '$'  DO
    BEGIN
    ch := token[1];
    IF  ch IN ['0'..'9','-']  THEN
      Push(token)
    ELSE IF  ch IN ['+','-','*','/','^']  THEN
      call(ch)
    ELSE
      Calculate(ch);
    token := outstr[i];  i := i + 1;
    END;
END;
```

โปรแกรมที่ 7 แสดงโปรแกรม Calculate เมื่อเขียนในแบบเรียกตนเอง



รูปที่ 3 แสดงโครงสร้างขั้นตอนวิธีของโปรแกรม DoStatement ในเรื่องภาษา IQ0 ซึ่งมีลักษณะเป็นต้นไม้ไม่รู้จบ ในการเลือกหนทางปฏิบัติที่ซ้อนๆกัน

คือ ถ้าอักขระตัวแรกของโทเค็นเป็นตัวเลข หมายความว่าโทเค็นนี้เป็นเลขจำนวน ให้เก็บลงสแต็ก (ซึ่งจะต้องแปลงจากสตริงของตัวเลขเป็นเลขจำนวนก่อน) โดยเรียกใช้ Push (ไม่ได้แสดงโปรแกรมนี้เช่นกัน)

แต่ถ้าเป็นเครื่องหมายในการคำนวณ ให้ทำการคำนวณโดยเรียกใช้ Call (ไม่ได้แสดงโปรแกรมเช่นกัน) ซึ่งจะทิ้งผลจากการคำนวณไว้บนชั้นยอดของสแต็กเสร็จแล้วให้อ่านโทเค็นเข้ามาอีก 1 ตัว ทำซ้ำอยู่เช่นนี้จนกว่าจะถึง $ ซึ่งปิดท้ายสตริงใน outstr

การดำเนินการแบบนี้ใช้ได้กับข้อมูลแบบที่ 1 และ 2 แต่เราต้องการให้ทำกับข้อมูลแบบที่ 3 ด้วย คือเมื่อมีตัวแปรมาร่วมด้วย เช่น ตามตัวอย่างที่ยกมาคือ 39 – B8 + 75 * C5 ขอให้พิจารณาดูว่าจะทำได้อย่างไรครับ เพราะตัวแปรก็คือชื่อเซลล์และในเซลล์หนึ่งๆอาจจะเป็นข้อมูลแบบใดแบบหนึ่งในสามแบบที่กล่าวมา และอาจจะซ้อนๆต่อไปอีกกี่ชั้นก็ได้ ดังแสดงเป็นโครงสร้างข้อมูลได้ดังรูปที่ 2

ซึ่งอาจกล่าวได้ว่า *หากเราเห็นว่าโครงสร้างข้อมูลเป็นแบบต้นไม้ (tree) ไม่รู้จบเช่นนี้ ก็เป็นที่แน่ใจได้ว่าเหมาะสำหรับโปรแกรมเรียกตนเอง* ดังแสดงในโปรแกรมที่ 7 ซึ่งถ้าเขียนในแบบธรรมดาแล้วจะเขียนได้ยากมาก

อีกแบบหนึ่ง ต้องขอให้ดูเรื่อง **"ภาษา IQ0"** (ไมโครฯ 35) ในโปรแกรมย่อย DoStatement ซึ่งเป็นโปรแกรมที่จะแปลประโยคในภาษา IQ0 เป็นรหัสเครื่อง ซึ่งในภาษา IQ0 ได้กำหนดรูปแบบของประโยคไว้ 4 แบบ คือ

1. เป็นประโยคให้ค่าแก่ตัวแปร
2. เป็นประโยคคำสั่ง
3. เป็นประโยค IF
4. เป็นประโยค WHILE

ซึ่งในประโยค IF และ WHILE

จะมีประโยคทั้ง 4 แบบนี้กี่ประโยคก็ได้ และซ้อนกันกี่ชั้นก็ได้ ซึ่งสามารถเขียนเป็นโครงสร้างของขั้นตอนวิธีได้ดังรูปที่ 3 นั่นคือโปรแกรมที่เรามองเห็นว่าขั้นตอนวิธีเป็นแบบต้นไม้ไม่รู้จบ เหมาะที่จะเขียนในแบบเรียกตนเองเช่นกัน

จากที่กล่าวมานี้ในทัศนะของผมเห็นว่า งานที่ควรคือจำเป็นต้องเขียนโปรแกรมในแบบเรียกตนเอง คือ งานที่มีโครงสร้างข้อมูล หรือโครงสร้างขั้นตอนวิธีเป็นแบบต้นไม้ไม่รู้จบ หรืองานที่ต้องการผลการดำเนินกรรมวิธีครั้งก่อนมาดำเนินการใหม่ หรือดำเนินการต่องานเหล่านี้มักจะไม่พบในการเขียนโปรแกรมตามปกติ ส่วนมากจะพบในงานเหล่านี้

- การค้นหา (search) หรือการเรียงลำดับข้อมูล (sort) เช่นการเรียงลำดับข้อมูลในแบบควิกซอร์ต
- โปรแกรมตัวแปลภาษา หรือโปรแกรมที่อ่านข้อมูลของผู้ใช้เข้ามาตีความดังเช่นในเรื่องภาษา IQ0 และการเขียนโปรแกรมในยุคที่สาม
- ปัญญาประดิษฐ์ (AI) ดังเช่นที่กล่าวในเรื่องปาสคาลกับปัญญาประดิษฐ์ (ไมโครฯ 31) ที่รู้จักกันแพร่หลายได้แก่ โปรแกรมแก้ปัญหาหอคอยแห่งเมืองฮานอย

หมายเหตุ:

1. โปรแกรมที่ 6 และที่ 7 เป็นเพียงร่างโปรแกรมเพื่อประกอบคำอธิบายเท่านั้น ยังไม่ผ่านการทดสอบ

2. ที่กล่าวว่าเป็นแบบต้นไม้ไม่รู้จบนั้น ความจริงจะมีจุดจบในตัว เช่นในสเปรดชีตถูกจำกัดด้วย จำนวนตัวแปร ความยาวของสตริงในแต่ละเซลล์ และการใช้งานตามปกติ ส่วนในเรื่องภาษา IQ0 นั้น ถูกจำกัดด้วยการใช้งานตามปกติ คือ ผู้ที่เขียนโปรแกรมด้วยภาษา IQ0 จะใช้ IF และ WHILE ซ้อนๆ ไประดับหนึ่งเท่านั้น เช่นที่ใช้กันในภาษาปาสคาล

```
;-------------------------------------------------------------
stack       SEGMENT  stack 'STACK'
            dw       128 dup (?)
stack       ENDS
;-------------------------------------------------------------

cseg        SEGMENT  public  'CODE'
            ASSUME   CS:cseg, DS:nothing

main        PROC     far
start:

            call     Recur

terminate:  mov      ax,4C00h
            int      21h
main        ENDP
;.............................................................

Recur       PROC     near       ; BEGIN
            mov      ah,01h     ;   Read(KBD,ch); Write(ch);
            int      21h        ;
IF1:        cmp      al,'.'     ;   IF  ch = '.'  THEN
            jne      ELSE       ;
            mov      dl,'@'     ;     Write('@')
            mov      ah,02h     ;
            int      21h        ;
            jmp      ENDIF      ;
ELSE:                           ;   ELSE
            sub      al,20h     ;     BEGIN
            push     ax         ;       ch := UpCase(ch);
            call     Recur      ;       Recursion2;
            pop      ax         ;
            mov      dl,al      ;       Write(ch);
            mov      ah,02h     ;
            int      21h        ;
ENDIF:                          ;     END;
            mov      dl,20h     ;   Write(' ');
            mov      ah,02h     ;
            int      21h        ;
            ret                 ;
Recur       ENDP               ; END;
;.............................................................

cseg        ENDS
            END      start
```

โปรแกรมที่ 8 โปรแกรมเรียกตนเองในภาษาเครื่อง

ภาษาเครื่องกับการเรียกตนเอง

ภาษาเครื่องเป็นภาษาที่อ่อนตัวจะเขียนในรูปแบบใดก็ได้ จะเขียนให้ยุ่งพันกันเช่นภาษาเบสิกก็ได้ จะเขียนให้เป็นโมดูลเป็นโครงสร้างอย่างภาษาปาสคาลก็ได้ จะเขียนเป็นโปรแกรมเรียกตนเองก็ได้ (ถ้าเขียนไม่ได้ก็ไม่มีภาษาใดเขียนได้)

ผมได้นำโปรแกรมที่ 5 ซึ่งเขียนด้วยภาษาปาสคาลมาเขียนเป็นภาษาแอสเซมบลี ดังแสดงในโปรแกรมที่ 8 อย่างเรียกได้ว่าแทบจะตรงตัว ซึ่งโปรแกรมตัวแปลภาษาปาสคาลก็จะแปลออกมาในทำนองนี้เช่นกัน

ขอให้สังเกตการเก็บค่า al คือ push al ซึ่งก็คือการเก็บค่า ch ไว้ในสแต็ก (ของซีพียู) ก่อนการเรียกตนเองนั่นเอง และเมื่อส่งกลับมาแล้ว ก็จะมาถอนค่า al คือ pop al ซึ่งก็คือการคืนค่าให้แก่ ch นั่นเอง ดังนั้นเราจะมองเห็นเป็นว่าโปรแกรมสร้างตัวแปร ch เป็นจำนวนเท่ากับจำนวนครั้งที่ผู้ใช้กดคีย์อักษร

คิดว่าคงทำความกระจ่างให้แก่ผู้ที่สงสัยพอสมควรครับ อนึ่งสเปรดชีตที่ผมกล่าวถึงนี้จะตรงกับของจริงหรือไม่ ผมไม่ทราบด้วย เพราะผมไม่เคยใช้โปรแกรมพวกนี้